Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Token-Based Concurrency Control for Transaction Serialization in Replicated Database Systems over Decentralized P2P Networks

Katembo Kituta Ezéchiel

kkitutaezechiel@gmail.com

Institut Superieur de Commerce de Butembo (ISC/Butembo)

Research Article

Keywords: Token-based concurrency control, Transaction serialization, Mutual exclusion algorithm (MEA), Replicated Database System (RDBS), Peer-to-Peer (P2P) Network (key words)

Posted Date: November 9th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11272328/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

Token-Based Concurrency Control for Transaction Serialization in Replicated Database Systems over Decentralized P2P Networks

Katembo Kituta Ezéchiel
Associate Professor at the Département d'Informatique de Gestion,
Filière Géstion Commerciale et Administrative,
Institut Superieur de Commerce de Butembo (ISC/Butembo),
Butembo City, Congo(DRC)
kkitutaezechiel@gmail.com

***Abstract*—This research paper focuses on problems that occur when replicating the database (DB) over large-scale networks. In particular, the emphasis is on the decentralized Peer-to-Peer (P2P) architecture. The motivation is based on the trouble that arises when multiple processes need to simultaneously access the same resource (replica/copy). Often, updates made by refresh transactions from a peer group are performed concurrently on a destination peer. This phenomenon breaks the assurance of the proper execution of transactions and corrupts resource, or make it inconsistent. Therefore, this work proposes a token-based concurrency control approach to serialize refresh transactions performed concurrently by different peers on a copy of a Replicated Database System (RDBS). The algorithmic method and the circulating token ring topology made it possible to realize a distributed mutual exclusion algorithm for transactions serialization in a RDBS over a decentralized P2P network. After an embryonic performance analysis, the result is promising. (*Abstract*)**

Keywords—Token-based concurrency control, Transaction serialization, Mutual exclusion algorithm (MEA), Replicated Database System (RDBS), Peer-to-Peer (P2P) Network (*key words****)***

I. INTRODUCTION

Data replication is the only one strategy that ensures the consistency, improves data availability, provide good performance in terms of load balancing and response time in Distributed Databases Systems (DDBSs) [1]. This distribution strategy consists of storing replicas of the same DB in geographically distributed and computer networked sites and spreading updates from one or more sources toward one or more targeted destinations [1], [4]. The main objective of the replication is to maintain consistent replicas [2], [3]. Multi-master replication is one of replication protocols that offers good performance and availability benefits [4].

However, nowadays the P2P system applications advancement is emergent. Typically, they require general replication capabilities with different levels of granularity and multi-master mode [5]. In these kinds of systems, each peer can transfer updates to all others and several peers can update the same replica [6], [7]. Therefore, in a RDBS, refresh transactions serialization is a problem that deserves a special attention because they always conflict each other. Concretely, updates carried by refresh transactions, from different Peers reach a destination site at the same time and attempt to be performed on the same time. This concurrent execution always makes copies inconsistent [8].

The aim of this paper is to introduce a new mutual exclusion algorithm to serialize the distributed refresh transactions performed concurrently by different peers on a copy of a RDBS over a decentralized P2P network. Thus, to reach this purpose, the skeleton of this paper is organized as follow: the first section has introduced by presenting the context of this research as well as the status of the problem, the second section review the related works, the third present the proposed methodology, fourth section present the performance analysis and finally the fifth section concludes this study.

II. RELATED WORKS

In a RDBS environment, updates are data changes performed to a primary site and what need to be forwarded to secondary sites to always maintain the mutual convergence of replicas [8], [9]. Thus, running transactions is the popular technique used to achieve this. A transaction is a sequence of read or write operations performed to a Distributed Database Management System (DDBMS). It takes the DB from one consistent state to another consistent state and it ends with "commit", the validation of all operations or "rollback/abort", the cancellation of all operations made by it [11], [12]. In other words, forwarding updates is nothing else than reading new data changes from a master copy and writing them to slaves.

However, DDBMSs must ensure that transaction execution meets a set of properties that lead to the consistency of Distributed Databases (DDB) and conveniently summarized by Atomicity, Coherence, Isolation, Durability (ACID), since when the execution is always concurrent [11], [13]. Thus, several researches have already been undertaken to solve the transaction concurrency control problem. Concurrent execution without synchronization constraints produces a number of problems who's the most important are loss of operations and improper readings. Thus, it is necessary to fix a property determining a correct execution of transactions completion: the serializability [14]. The serializability is NP-Complete problem [15]. Transactions concurrent execution is correct (produces the same result) if it is equivalent to a serial running. Serialization is a strong property that limits parallelism to execution and improve performance [12].

However, in distributed computing systems several solutions have already been proposed to this problem of

concurrent and collaborative execution between multiple processes. Some approaches are those of serializing transactions performed in DDBMSs while others are those that comprehensively manage communication between nodes in a network by advocating mutual exclusion of processes to ensure sequential execution. Thus, first, we review some existing algorithms for the serialization of transactions in DDBMSs and thereafter this review will be extended to networks mutual exclusion approaches to ensure sequential execution.

A. Transaction serialization approaches

More than one transaction serialization algorithm, to attempt to achieve the serial order execution, exist and we can classify them by two tendencies [8], [15], [16], [22], [24]:

1) Pessimistic approach

It states that we cannot have a correct execution. We have found two algorithms: Two-Phase Locking (2PL) and Timestamp Ordering (TO) already implemented in most of commercial DDBMSs.

a) Two-Phase Locking (2PL) algorithm

In an industrial way, the only solution implemented is the locking approach. The 2PL algorithm is the oldest, and still the most used, concurrency control method ensuring strict serializability. Unfortunately, it is reputed to induce deadlocks as well as rejections of transactions [17], [18].

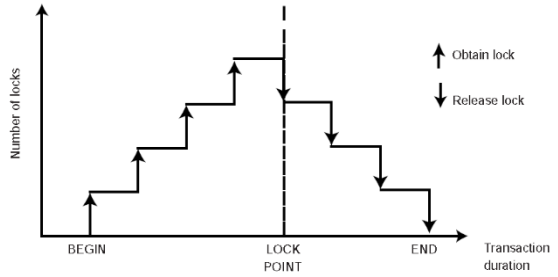


Fig. 1. Growing and shrinking phases in 2PL Protocol [8]

The 2PL is based on the locking of current read or write tuples. The idea is simple: as illustrated in the Fig. 1, each transaction wishing to read or write a tuple must first obtain a lock on this tuple. Once obtained, the lock remains held by the transaction that placed it, until this transaction decides to release the lock. A lock is a state variable associated with an object of the database and indicating its state with respect to read/write operations [12], [13].

b) Timestamp Ordering (TO) algorithm

In order to maintain transactions serialization in a DDBSs, apart from the Locking-based algorithm, there is a Timestamp-based algorithm. With this technique, to maintain the execution of transactions in serial order, on the initialization of transactions, the transaction manager assigns to each one a unique timestamp for its identification and for transactions ordering [17], [18].

The main rule is enunciated as follow: suppose two conflicting operations, which can be read or write SQL commands in a transaction such that O_{ij} and O_{kl} respectively belonging to the transactions T_i and T_k, and ts, the Timestamp. O_{ij} is executed before O_{kl} if and only if $ts(T_i) < ts(T_k)$; in other words, T_i would be the oldest transaction and T_k would be the youngest. This protocol avoid completely *deadlocks* of transactions because even if $ts(T_i)<ts(T_k)$, the transaction T_k cannot *rollback* rather the operation O_{kl} will be rejected and the transaction manager would restart the whole transaction with a new Timestamp [8].

The visible limitation of Timestamp Ordering (TO) technique is that several restart of transactions influence negatively the performance of the system. There are other algorithms to attempt to improve this TO technique: Conservative TO and Multi-version TO algorithms which aims to reduce the number of transaction restarts [8], [15], [16]. However, when one uses timestamp technique in locking-based algorithm in order to improve the concurrency level and efficiency, it emerges the Hybrid technique.

c) Hybrid technique

This technique combines the advantages of 2PL algorithm and TO algorithm, in other word the notion transaction lock as well as the transaction timestamp. According to [8], this still being a challenge, since this has never been implemented in any commercial DDBMS. But some researches have already been proposed in this way. So, [12], [13], [18] present a hybrid algorithm: Wait-Die (WD) &Wound-Wait (WW). This algorithm follows 2PL technique principals but overcome deadlock problem by applying the TO technique rule.

2) Optimistic approach

It executes the transaction without constraints and at the time of the validation one verifies if there are no conflicts with the other transactions. The optimistic approach of the concurrency control technique is appropriate in low conflict systems because the validation of each transaction for serialization, can reduce performance. In these cases, the serialization test is adjourned just before validation because the conflict rate is low and the probability of aborting no serializable transactions is low as well [8].

Until today no research has yet attempted to implement the 2PL technique for the optimistic approach [15]. But according to [8], it would be possible to design a technique lock-based for this approach. Some of the proposed algorithms are timestamp-based and some have already been extended to DDBMSs [16]. One of them is Distributed Optimistic Protocol (OPT) [12], [13], [17], [18]. It is a time-based concurrency control algorithm that works by exchanging certification information during the commitment. For each data item, a read timestamp and a write timestamp are retained. OPT can provide the possibility of serialization under the restrictions imposed by two timestamps read and write basically assuming here above. If the concurrency check is not used, non-serializable execution orders can be generated by concurrent transactions.

Apart from these two trends, there is also a network technique for mutual exclusion to ensure sequential execution between processes. It is necessary to review this technique to understand how it works, as its improvement turns out to be another way to ensure efficient serialization of transactions.

B. Mutual exclusion technique for sequential execution

This section clarifies the uniqueness problem and presents an overview of the generic solution provided to the mutual exclusion problem on the distributed architecture of a network. However, we start with the key concept “critical section”. In concurrent programming, a section is said to be critical if its access by concurrent processes or threads results

in an inconsistent state. Usually, it results from a shared resource. At any one time, at most one process is allowed to activate in this section [19].

For this problem, the following requirements are set to ensure the sequentiality or serializability of operations in a critical section [20]:

1. At a time *t*, a single process occupies the critical section;

2. No order of priority is assigned to the processes, just as the relative speed of the processes is not a decisive factor for entering the critical section;

3. Stopping or suspending any process in the system must, in no case block other processes;

4. At the simultaneous request of several processes, the final selection must be done in a finite time.

By these rules, the atomicity of operations on a resource in a critical section is guaranteed. We then speak of execution in mutual exclusion. In this context, the keyword "lock" is also often used. It reflects the notion of critical section associated with an instruction block, in which a particular object is affected by the locking. Physically, mutual exclusion is achievable if interruptions are prohibited while a shared resource is modified within the critical section [21].

1) Properties of mutual exclusion

The mechanisms of synchronization between processes and mutual exclusion have been adopted in both parallel and distributed environments long time ago. Different in their architecture, these environments have in common that they have concurrent processes. Thus, according to [19] and [20], any algorithm that claims to ensure mutual exclusion must respect the following properties:

Safety: nothing bad happens. No two processes can share the critical section.

Liveliness: any process requesting a critical section must be able to access it after a finite amount of time.

The algorithm has to protect itself against famines and ensure fairness between processes. An algorithm is very fair if it respects the order of arrival of the processes in decision-making. To achieve this, a queue is made available to the processes to serve them according to the FIFO (First In/First Out) order. While low equity ensures that there is no famine, it nevertheless admits that other privileged processes, which enjoy higher priority, outstrip one process.

2) Deadlocking in distributed environments

Deadlock is the worst thing that can happen in concurrent programming. This state reflects two mutually pending processes. Processes stuck in this state are permanently stuck [19], [22], [24]. In the literature, we have not found an effective solution to this problem, because when a process holds a resource in critical section, no one can force it to release it. In a system with *I* processes, the interlocking takes the form of a cycle in a wait graph according to the following diagram: Let $P = \{p_0, p_1...p_i\}$, a set of processes such as: p_0 waits p_1 waits ... p_{i-1} waits p_i waits p_0. Thus, to avoid interlocking in a system with *i* competitive processes, the wait graph must be acyclic. Among the possible strategies for an algorithm to be free from this problem, we can cite the deadlock prevention, deadlock avoidance and deadlock detection [19], [22].

3) Taxonomy of distributed mutual exclusion algorithms

A very wide variety of mutual exclusion algorithms exist to provide some solution to the growing problems of resource sharing in distributed systems. The dissimilarity of distributed environments and the paradigms exploiting them are major factors leading to the evolution of these algorithms. These algorithms immediately followed the progression of architectures, to consider those based on sending messages as well as those with shared memory.

According to [23] mutual exclusion algorithms are categorized into two broad families: permission-based and token-based ones. However, it exists very sparingly in the literature. In this research, we are interested in the second family of algorithms, those based on the passing of a token, because of qualities offered by them. Given that, the existing ones were developed not having from the outset the idea of the serialization of the transactions in a RDBS, for what concerns this research, we review only the generic algorithm of their category.

4) Circulating token ring technique

In computer networking the circulating token ring technique uses the "Token Ring" communication protocol which defines the sequential order of sending messages by nodes or processes [24], [25], [26]. This protocol is based on the principle of turn-by-turn communication, i.e. every node in the network has the opportunity to speak in its turn. Nodes are connected to each other by means of the same ring [24], [26]. This protocol uses a special packet of data called "token" that moves around a ring in a loop from one node to another to determine which computer has the right to issue information. When a node is in possession of the token it can issue for a specified time, after which it hands over the token to the next node [24], [27]. In this way, the token as well as the packet of data circulate around the ring to reach the node, which is the destination, as illustrated in the Fig. 2.

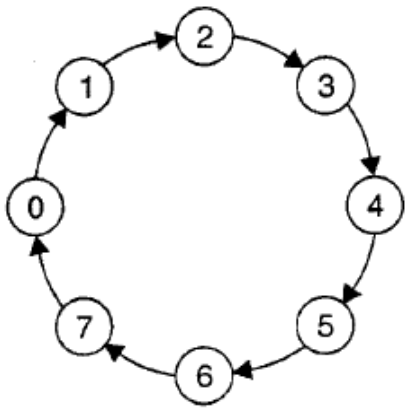


Fig. 2. A logical ring constructed in software [24], [26].

However, when the ring is initialized, the node 0 receives the token. Thus, in general way, when any node acquires the token from its predecessor, it acknowledges receipt and subsequently checks whether it needs to access the critical section in order to manipulate the shared resource. If so, then it does what is necessary, frees the resource, and passes the token to its successor in the ring. As soon as the token is released to the neighbour, the node is no longer allowed to

perform an operation on the resource using the same token. If a node receives the token from its predecessor and is not interested to enter the critical section to use the resource, it simply passes the token to its successor. Therefore, when no node needs to enter the critical section, the token simply circulates around the ring. So, if one can imagine the case where a node p_i crashes, it will therefore be necessary for the node p_{i-1} to give the token to the node p_{i+1} because when it tries to give it to p_i it would not acknowledge receipt.

Obviously, this algorithm is better for a stable network whose nodes are always available and their configurations are homogeneous in advance. But the difficulty remains possible. Once a node decides to enter the critical section, at worst, it will have to wait until all the other nodes access the critical section so that it finally receives the token at the end. In other words, when a node releases the token it automatically becomes the last in the list. Another problem arises when the token is lost; it should normally be regenerated. However, detecting the loss of the token is difficult since the time of the token's turn on the network is unlimited. Just because the token has not been spotted for any length of time does not necessarily mean that it is lost. A node that is still using it can still hold it.

This algorithm also runs into trouble if a process crashes. The recovery is easier than in the other cases because it is required that a process receiving the token acknowledge receipt. So a dead process is detected when its neighbour tries to give it the token and fails. At that point the dead process can be removed from the group, and the token holder can throw it over the head of the dead process to the next member down the line, or the one after that. Doing so requires that every node maintain the current ring configuration. Still, in the P2P network, the nodes do not always have the same configuration. Hence the need to review this topology for its adaptation to the characteristics of the P2P networks.

III. METHODOLOGY

A RDBS on a P2P network includes *p* peers and each peer has a copy of the database. It can leave or join the network anytime and its configuration is not necessarily same with that of others. Peers communicate with each other by exchanging messages and forwarding updates by *transactions*. Therefore, transactions run according to a *new circulating token algorithm*, which determine their serialization order. This approach provides not only a new algorithm for managing a circulating token ring topology that effectively take in consideration characteristics of P2P networks, but it also solves the problem of the worst time complexity when a process decides to enter the critical section as it has to wait for all other process to use the resource before it also get the access on it. The other problems are those of token loss and that of process crash. Moreover, it also provides a solution to the problem of concurrency control between update transactions.

The algorithms of two trends reviewed in the literature, namely the pessimistic and optimistic tendency, have not been up to solve effectively the concurrency control problem. So, to hope to solve this problem using the ring topology with circulating token, the worst time complexity when a process decides to enter the critical section, the loss of the token and the unavailability (crash) of the process deserve an effective solution. Finally, the algorithm requires an adaptation that can satisfies on the characteristics of P2P networks to work better. Thus, with this new approach, it can also be given the privilege to updates from certain sites considered more important.

A. New circulating token ring network topology

Assume a network consisting of 8 peers all networked. The Fig. 3 below presents the decentralized P2P circulating token ring network topology for a RDBS.

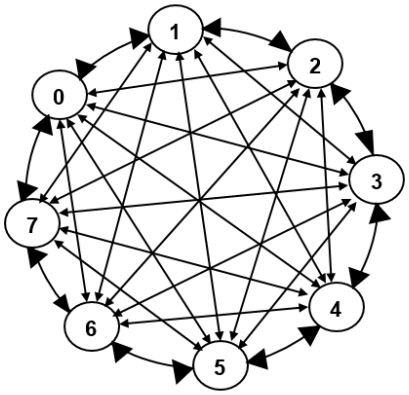


Fig. 3. Decentralized P2P circulating token ring network topology.

In this new approach, peers of a "token ring" type network are not arranged in a loop, but are virtually connected to a splitter called Multi-station Access Unit (MAU) which gives successively "the word" to each of them [26], [27]. Reason why, not as it is always done in the everyday circulating token ring topology, in this case, the predefined order of getting or releasing the circulating token is not necessary, because we deal with a P2P network where a peer can or cannot be available. So, the optimization policy is to give the token directly to a peer which needs it instead of going through a ring of peers while attempting to release the token to the next peer. The probability that it be unavailable when of the token release is of ½ or 50%; where *1* is the next peer in the list and 2 the possibilities (presence/absence) for it to be or not to be available when its predecessor wants to hand over to it the token.

As the token is expressed in bit, this method minimizes the "Ring Latency (RL)", the time taken by a bit of the token to go through the ring [26]. Typically, the RL is obtained by the formula : $RL= d/v+p*t$, where d is the length of the ring, v is the velocity of data in ring, p is the number of peers in ring and t is the time taken by each peer to hold the bit before transmitting it (bit delay). The term d/v is the propagation delay (in sec) and $p*t$ is the bit delay (in bits) [27].

Proof1: Let P = {p_1, p_2 ,. . .p_n} be the set of processes, i.e. set of participating peers such as cardinal of P, noted #P ≥ 2. Suppose that #P = n and that p_i, an element of P wants to release or hand over the token (for ex. to p_j as it needs it). It will first release it to p_{i+1} and p_{i+2} to p_{i+3}… and p_{j-1} to p_j. If p_{i+1} up to p_{j-1} are not available or do not need to enter the critical section, then it will take a time T_r (RL= T_r, *the token release time*). T_r is obtained for the firt time by the calculation T_{r1} = t_{i+1} + t_{i+2} + … + $t_{j-1,}$

$$T_{r1} = \sum_{i=1}^{j-1} t_i \quad (1)$$

the time which must pass while some of these j-1 peers are passing the token to the next one if it is available or to the next to the next one if the next one is unavailable until to reach p_j that is available and needs the token.

Secondly, at the time of the token release by p_i, suppose that p_j had already felt the need to get it. However, during its passage from p_{i+1} to p_{j-1}, if some peers also had this same need, then they will use the token before it reaches p_j even though p_j wanted it before them. In this logic, the time T_r the token can take to reach p_x is calculated as follows: $T_{r1} = (\varepsilon + t_{i+1}) + (\varepsilon + t_{i+2}) + \ldots + (\varepsilon + t_{j-1})$,

$$T_{r1} = \sum_{i=1}^{j-1} \varepsilon + t_i \quad (2)$$

where "ε" is the time that a peer p_i in the sequence from p_{i+1} to p_{j-1} may have taken in accessing the critical section before p_j when it even felt the need to get the token after it.

Proof2: Based on the illustration given above, suppose that p_i needs the token and the token is owned by p_{i+1}. With #P = n, the time T_g (in this case RL= T_g, *the token getting time*) will be obtained at the first time by $T_{g1} = t_{i+1} + t_{i+2} + \ldots + t_n + t_1 + t_2 + \ldots + t_{i-1} + t_i$.

$$T_{g1} = \sum_{i=i+1}^{i} t_i \quad (3)$$

We can also add the following possibility: among those peers to whom the token must circulate, while p_i has felt the need to get it before them, a certain peer p_x may also feel its need even after p_i. Well in this case, when the token is in its possession, it will take a more time "θ" for it to access the critical section before releasing the token. Hence, the time T_g (RL) deviates $T_{g1} = (\theta + t_{i+1}) + (\theta + t_{i+2}) + \ldots + (\theta + t_n) + (\theta + t_1) + \ldots + (\theta + t_{i-1}) + (\theta + t_i)$,

$$T_{g1} = \sum_{i=i+1}^{i} \theta + t_i \quad (4)$$

with θ the probable time that a peer p_x of this suite may have experienced the need of the token after p_i, may take to enter the critical section before releasing the token.

Correctness: These anomalies deserve a method, which minimizes the time $T_{(r\ \&\ g)}$ i.e. the RL. That is why to manage the two crucial events, namely releasing or handing over and getting the token, in order to serialize efficiently refresh transactions in P2P replication, the algorithms have been designed.

1. With this new approach, in the set P so the #P = n, the token is released directly to p_j as the one that needs it on time $T_{r2} = t_i$ and p_j will give it to p_x that will need it after him on another time t_i; then $T_{r2} = t_i$ and so on.

$$T_{r2} = t_i \quad (5)$$

Only two conditions are observed: in the best case, p_j has to be the first privileged/unprivileged peer and its request time less than those of others requesting peers. In the worst case, p_j can be a last privileged/unprivileged peer and its request time greater than that of others. So, the running time becomes

$$T_{r2} = \mu + t_i \quad (6)$$

where "μ" is the duration that elapses when testing alternatives privileged/unprivileged and on which time requesting peers requested the token. So, T_r deviates minimized because $T_{r2} < T_{r1}$.

2. Similarly, to get the token, in the best case, it takes a time $T_{g2} = \omega + t_i$ because p_j will has to find the token directly at p_{j+1} and place its request directly there.

$$T_{g2} = \omega + t_i \quad (7)$$

In the worst case the token being at p_{j-1} there will be a time $T_{g2} = (t_{j+1} + t_{j+2} + \ldots + t_n + t_1 + t_2 + \ldots + t_{j-2}) + (\omega + t_{j-1})$,

$$T_{g2} = \left(\sum_{i=j+1}^{j-2} t_i \right) + \left(\omega + t_{j-1} \right) \quad (8)$$

in all cases with ω the time that p_j places its request at p_{j+1} or p_{j-1} and p_{j+2} or p_{j-1} hand-over the token to p_j. However, the time ω, is a duration that elapses when p_j is introducing the token request to p_{j-1} or to p_{j+1}. So, T_g is minimized because $T_{g2} < T_{g1}$.

Thus, before presenting our entire approach in a purely algorithmic language, the following management standards allow us to specify the new model of data and processing:

1. A P2P replication system updates all databases of all participating peers. Before running the synchronization for the first time, the administrator must first initialize the system;

2. To enter the critical section in order to update databases of the participating peers, a coordinator peer must have the token conferring to it the status of the master. And all others at that moment become slaves. The flow of this token on the topology is managed by specific methods, algorithm 1, 2, 3 and 4, which successively determine available peers, how a peer can get the token and how it can realise it;

3. To get the token, a peer must send a request on the network to identify the peer that owns it. As soon as found, it submits a token request to the current token owner. Else, it sets its status as token owner because if no peer owns it, it is directly supposed that the last owner has released it when no one did not request it or it goes down or crashes;

4. Once a peer gets the token, it can run the synchronization. This synchronization consists of spreading its updates to other peers. These updates consist of all write transactions executed by the user's application;

5. When the synchronization process ends, the token is given to another participating peer, which has requested it, or it is released if any peer has not requested to the current owner. Among the peers that have requested, the token is given to the one which has a privilege and whose date and time of application is minimum or old among those of the privileged applicants. Otherwise, if no privileged peer has requested, the

token is given to an unprivileged peer whose date and time of request is minimum or old among the non-privileged.

These standards are of paramount importance for implementing this new approach. Nevertheless, although in this work the study is limited to the analysis of the theoretical temporal complexity. The new approach is illustrated fully algorithmically.

B. New circulating token ring algorithm

It is already known that $P = \{p_1, p_2, \ldots p_n\}$, is the set of processes, i.e. the set of participating peers such as cardinal of P, noted $\#P \geq 2$. Elements of the set P are divided in 2 subsets: privileged peers, noted $P^P = \{p^p_1, p^p_2, \ldots p^p_n\}$ and unprivileged peers, noted ${}^UP = \{{}^up_1, {}^up_2, \ldots {}^up_n\}$ so that $P^P \cup {}^UP = P$. Let also suppose that P^A is the set of available processes such as $\#P^A \leq \#P$ and P^U is the set of unavailable processes such as $\#P^U < \#P$. A peer p^a_i is available if and only if its connection returns a "true" value; else, it is noted p^u_i i.e. unavailable peer. We can therefore define the sets $P^A = \{p^a_1, p^a_2, \ldots p^a_n\}$, $P^U = \{p^u_1, p^u_2, \ldots p^u_n\}$ and denote that $P^A \subset P$; $P^U \subset P$ and $P^A \cup P^U = P$. The RDBS on a decentralized P2P network is defined as follows: let $C = \{c_1, c_2, \ldots c_n\}$ be a set of copies of the completely RDB and respectively on the peers of the set P. For each $p_i \in P^A$, p_i can change the state at a given moment: it can be denoted by p_m, a master peer, when its token possession status variable has a "true" value; otherwise, it is noted s_i and qualified as a slave peer. So then, $\exists\, s_i \in P^S$, $P^S = \{s_1, s_2, \ldots s_n\}$ is a set of slave peers, therefore peers whose token possession status variable contains the "false" value. We can therefore denote $P^S \subset P^A$ and $\{ p_m \} \cup P^S = P^A$. Table 1 summarizes key notations used in this modeling.

Table 1: Summary of key notations.

Symbol	Meaning
P	a set of peers (processes or nodes)
p_i	a peer belonging to the set P
P^P	a set of privileged peers
p^p_i	a privileged peer
${}^UP$	a set of unprivileged peers
${}^up_i$	an unprivileged peer
P^A	a set of available peers
p^a_i	a peer belonging to the set P^A (an available peer)
P^U	a set of unavailable peers
p^u_i	a peer belonging to the set P^U (an unavailable peer)
p_m	a peer belonging to the set P but owning/receiving the token (a master peer)
P^S	a set of slave peers
s_i	a peer belonging to the set P^S (a slave peer)
C	a set of copies of the DB replicated over the P2P network
n^UP	number of unprivileged peer
nT_q	number of token request

Thus, the serialization of transactions is managed by these new procedures: algorithm 1 for identifying available peers.

Algorithm 1. P2P identifying available peers

Input: P: a set of peers (processes or nodes)
Output: P^A : a set of available peers
begin
1: **select * from** *P (all peers)*
2: **for** *(i ← 0* **to** *#P – 1)* **do**
3: **if** *(p_i.ConnexionStatus = "True")* **then**
4: p_i.Availability ← "True"
5: **else**
6: p_i.Availability ← "False"
7: **end if**
8: **end for** p_i
9: **return** P^A *a set of available peers*
end

After having a set/list of available peers i.e. connected processes or nodes, the following procedure is to get the token from them, thus the algorithm 2.

Algorithm 2. P2P getting the circulating token

Input: P^A : a set of available peers
Output: a Token request success message or a Token possession "True" status
begin
1: **select * from** P^A *(all available peers)*
2: **for** *(i ← 0* **to** *#P^A – 1)* **do**
3: **select** TokenPossessionStatus **from** p^a_i
4: **if** *(p^a_i.TokenPossessionStatus = "True")* **then**
5: send TokenRequest to p^a_i
6: **return** *a Token request success message* ***(end for** i)*
7: **end if**
8: **if** *(i = #P^A – 1)* **then**
9: p_m.TokenPossessionStatus ← "True"
10: p_m.TokenReceptionDateTime ← now()
return *a Token possession "True" status*
11: **end if**
12: **end for** *i*
end

As soon as a peer of the ring gets the token it takes the status of the master (p_m). Then it can execute the refresh transactions. As a result, all of them performed by p_m are accepted because it is the only one to propagate its updates at the time. Therefore, there is no rejection due to conflicts between refresh transactions. p_m possessing the token has the permission to execute a transaction to broadcast updates and bring the data of other elements of the set P^A to the convergence. If p_m own the token, as soon as it finishes spreading its updates to other p_s, it releases the token and after placing a token request at p_m any p_i may get it in a random manner. For an any $p_i \in P$ to be a potential candidate to receive the token form p_m, its request must have the minimum date and time of all requests received by p_m; or it must have a privilege but also still have the date and time minimum of all requests of privileged peers (p^p_i) received by p_m. In the latter case, if there is no peer that requested the token, p_m must release it so that when a peer wants to propagate its updates take it, according to the instructions of the algorithm 3 and 4.

Thus, algorithm 3 here below, describes the token release process.

Algorithm 3. P2P releasing the circulating token

Input: $P^A - p_m$: set of available peers except the master
Output: nothing
begin
select * from *P^A (all available peers)* **except** p_m
$n^UP \leftarrow 0$
$nT_q \leftarrow 0$
for *($i \leftarrow 0$* **to** *$\#P^A - 1$)* **do**
select *TokenRequest* **from** s_i
if *(s_i.TokenRequest = "True")* **then**
$nT_q \leftarrow nT_q + 1$
if *($n^UP > 0$)* **then**
if *(s_i.PrivilegeStatus = "True")* **then**
if *(s_i.TokenRequestDateTime = Min(s_i.TokenRequestDateTime))* **then**
//call P2P update releasing the circulating token procedure
updateReleasingToken(p_m, s_i)
end for *i*
else
Continue *($i \leftarrow i + 1$)*
end if
else
if *($i = \#P^A - 1$)* **then**
if *(s_i.TokenRequestDateTime = Min(s_i.TokenRequestDateTime))* **then**
//call P2P update releasing the circulating token procedure
updateReleasingToken(p_m, s_i)
end for *i*
else
$n^UP \leftarrow n^UP + 1$
$i \leftarrow -1$
end if
else if *($i < \#P^A - 1$)* **then**
Continue *($i \leftarrow i + 1$)*
end if
end if
else
if *(s_i.TokenRequestDateTime = Min(s_i.TokenRequestDateTime))* **then**
//call P2P update releasing the circulating token procedure
updateReleasingToken(p_m, s_i)
end for *i*
end if
end if
else
if *($i = \#P^A - 1$)* **then**
p_m.TokenPossessionStatus ← "False"
p_m.TokenReleaseDateTime ← now()
end if
end if
end for *i*
end

For the best readability and the concision of the algorithm 3, there is a procedure as subroutine, the algorithm 4 to update releasing the circulating token returning a Token realising success message.

Algorithm 4. P2P update releasing the circulating token

Input: p_m : a peer belonging to the set P but owning the token (a master peer)
s_i : a peer belonging to the set P^S (a slave peer)
Output: a Token realising success message
begin
p_m.TokenPossessionStatus ← "False"
p_m.TokenReleaseDateTime ← now()
s_i.TokenPossessionStatus ← "True"
s_i.TokenRequest ← "False"
return *a Token realising success message*
end

The token ring network topology having been combined with the decentralized P2P architecture has made it possible to achieve the new circulating token ring technique adjusted to the characteristics of P2P networks. This new approach is designed for serializing refresh transactions. What should follow in the next lines is to analyse the performance of this new approach and to compare it with existing ones.

IV. Performance analysis

This performance analysis aims to present the following two aspects:

(1) The comparative study of the Ring Latency (RL) of the new approach (P2P Distributed Mutual Exclusion Algorithm "P2P-DMEA") with the existing Token Ring Algorithm "TRA". Starting from a simple random sampling, we would like to illustrate the worst and the best cases of the RL of the two approaches;

(2) The performance of the new transaction serialization algorithm (P2P-DMEA) comparatively to the Distributed Two-Phase Lock (D2PL) currency control implemented in the most of DBMSs.

Let us assume a P2P network consisting of 12 peers and a random sample of token release and token getting times in milliseconds (ms) presented in the Table 2 as follows:

Table 2: Token release and getting time sample data.

Obs.	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
$\mathbf{t_i}$	8	8	8	8	8	8	8	8	8	8	8	8
ε	8	9	10	8	6	5	4	9	10	5	4	3
μ	2,5	2,5	2,5	2,5	2,5	2,5	2,5	2,5	2,5	2,5	2,5	2,5
θ	7	13	19	25	5	9	18	13	12	18	11	10
ω	0	0	0	0	0	0	0	0	0	0	0	12

From this sample presented in Table 2 above, it is first presented the comparison of the Ring Latency (RL) i.e. the efficiency between the P2P-DMEA and the TRA based on the mathematical models that successively determine the token release and the token getting time.

A. Efficiency of the circulating token ring topology

Two models of the token release (Tr) and the token getting (Tg) time have been successively compared:

- Tr_1: models 1 (for the best case) and 2 (for the worst case) and Tr_2: models 5 (for the best case) and 6 (for the worst case);
- Tg_1: models 3 (for best case) and 4 (for the worst case) and Tg_2: models 7 (for the best case) and 8 (for the worst case).

To carry out the efficiency analysis, based on the calculation of the execution time according to the sample data presented in Table 2 above, the execution times are presented and plotted successively in the Table 3 and in the graphs of Fig. 4 as follow:

Table 3: Results of the efficiency analysis based on the models of the execution time.

Obs.	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Tr_1 (1)	8	16	24	32	40	48	56	64	72	80	88	96
Tr_1 (2)	10	27	45	61	75	88	100	117	135	148	160	171
Tr_2 (5)	8	8	8	8	8	8	8	8	8	8	8	8
Tr_2 (6)	10,5	10,5	10,5	10,5	10,5	10,5	10,5	10,5	10,5	10,5	10,5	10,5
Tg_1 (3)	8	16	24	32	40	48	56	64	72	80	88	96
Tg_1 (4)	16	40	56	72	88	104	120	136	152	168	184	200
Tg_2 (7)	8	8	8	8	8	8	8	8	8	8	8	20
Tg_2 (8)	8	16	24	32	40	48	56	64	72	80	88	108

All the basic factors remaining unchanged, the models of the token release time (models 5 and 6) and the token getting time (models 7 and 8) for the new P2P-DMEA prove to be efficient.

Looking at the results obtained in the Table 3, after submitting the sample of data presented in Table 2 to models (equations) 1 to 8, the observation is that the RL of the TRA is increasing. Thus, when the number of peers increases, the RL also increases. This can be seen from lines Tr_1 (1) and Tr_1 (2) for the token release and lines Tg_1 (3) and Tg_1 (4) for token getting in Table 3.

While with the new P2P-DMEA the RL is identical regardless of the number of peers. This is described from lines Tr_2 (5) and Tr_2 (6) for token release and Tg_2 (7), of the same Table, for token getting, for the best case. However, Tg_2 (8) for the worst case of token getting, the RL is increasing but lower than that of the worst case of the token getting of the TRA (Tg_1 (4)). This same analysis can be read in the graphs of the Fig. 3.

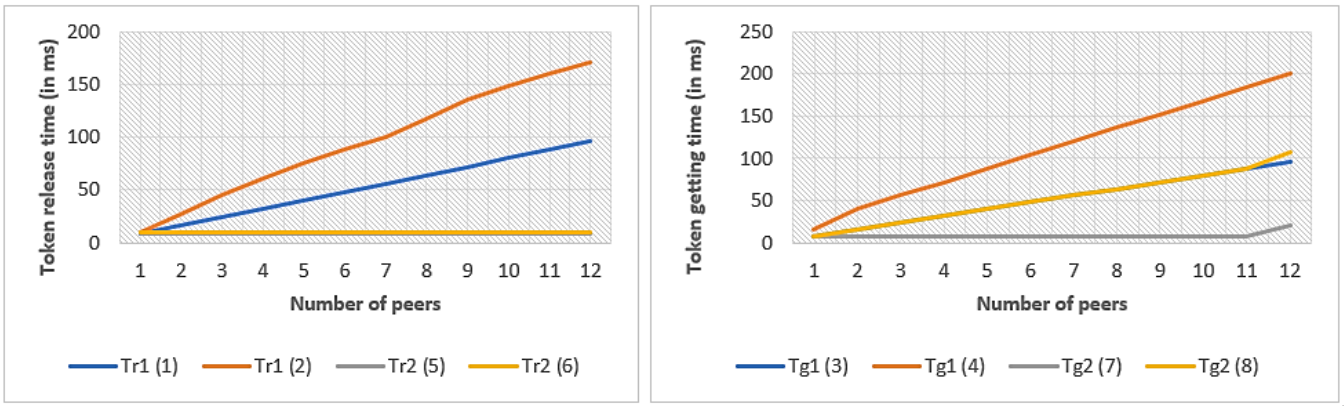


Fig. 4. Effectiveness of the token release (*left*) and the token getting (*right*) for the new P2P circulating token ring topology.

The illustrative curves of the token release and token getting time in the TRA are growing while those of the P2P-DMEA are almost parallel to the abscess axis, i.e. the token release time and the token getting time are equivalent for all peers. Except the curve of the token getting in the worst case for the new P2P-DMEA. Even if Tg_2 (8) is increasing but it is not higher than Tg_1 (4), the worst case of the token getting for the TRA. On these graphs of the Fig. 4, this is clearly observable with the different illustrative curves: for all cases, visibly $Tr_2 < Tr_1$ and $Tg_2 < Tg_1$.

Subsequently, this study must be extended to the comparative analysis of the theoretical time complexity of the algorithm.

B. *Efficiency of the circulating token ring algorithm*

To analyse the efficiency of an algorithm, the experimental method is frequently used. However, although valuable, experimental analysis has limitations because it does not allow analysing a particular algorithm without performing experiments on its execution time. Thus, in this work, the more analytical approach consisting in counting the high-level primitive operations of the pseudocode has been adopted, independently of the implementation programming languages. This method consists of a model of a Random-Access Machine (RAM) [28].

A primitive operation corresponds to a level instruction with an execution time that depends on the hardware and software environment but is constant. Instead of trying to determine the specific execution time of each primitive operation, we simply count the number of primitive operations executed and use the number *t* as a high-level estimate of the algorithm's execution time.

This number of operations has been correlated to a real execution time in a specific hardware and software environment, because each primitive operation corresponds to a constant-time instruction, and there is only a fixed number of primitive operations. The implicit assumption in this approach is that the execution times of the different primitive operations are identical. Therefore, the number, *t* of primitive operations that our algorithms perform is proportional to the real execution time of these algorithms.

1) The new P2P-DMEA complexity models

The new P2P-DMEA has two phases: the circulating token getting phase, executed by algorithms 1 and 2, and the circulating token releasing phase executed by algorithm 3, which in turn calls the algorithm 4. Between these 2 phases the processes owning the token can accesses the shared resource or data.

Here are the time complexity models for the new P2P-DMEA, circulating token getting phase:

- Algorithm 1 (P2P identifying available peers): O(n), where n is the number of peers (processes or nodes)
- Step 1: O(n) to select all peers
- Steps 2-8: O(n) to iterate over the peers and check their connection status
- Algorithm 2 (P2P getting the circulating token): O(n), where n is the number of available peers
- Step 1: O(n) to select all available peers
- Steps 2-12: O(n) to iterate over the available peers and check their token possession status

Here are the spatial complexity models of the new P2P-DMEA, circulating token getting phase:

- Algorithm 1: O(n), storing available peers
- Algorithm 2: O(1), storing the token

Total complexity for the token getting phase:

- Time complexity: $O(n) + O(n) = O(2n) \approx O(n)$
- Spatial complexity: $O(n) + O(1) = O(n)$

The total complexity of algorithms 1 and 2 is therefore O(n) in terms of time and space.

Here are the time complexity models for the new P2P-DMEA, circulating token release phase:

- Algorithm 3 (P2P releasing the circulating token): $O(n^2)$, where n is the number of available peers
- Step 1: O(n) to select all available peers except the master (since it holds the token)
- Steps 4-41: O(n) to iterate over the available peers and check their token request status in order to release the token to one of them according to the success of different conditions
- Steps 11, 19, 31: O(1) to call the updateReleasingToken procedure (algorithm 4)
- Algorithm 4 (P2P update releasing the circulating token): O(1), constant operation

Here are the spatial complexity models of the new P2P-DMEA, circulating token release phase:

- Algorithm 3: O(n), storage of available pairs and token queries
- Algorithm 4: O(1), storage of the token

Total complexity for the token releasing phase:

- Time complexity: $O(n^2) + O(1) = O(n^2)$
- Space complexity: $O(n) + O(1) = O(n)$

The total complexity of algorithms 3 and 4 is therefore $O(n^2)$ in terms of time and O(n) in terms of space.

Thus, Table 4 presents the models obtained after analysing the theoretical time and space complexity of the new P2P-DMEA.

Table 4: Models of the theoretical time and space complexity of the new P2P-DMEA.

Algorithm phases	Time complexity	Space complexity
1	*O(n)*	*O(n)*
2	*O(n)*	*O(1)*
Total complexity token getting phase	$O(2n) \approx O(n)$	O(n)
3	*$O(n^2)$*	*O(n)*
4	*O(1)*	*O(1)*
Total complexity token releasing phase	$O(n^2)$	O(n)

2) The D2PL complexity models

Likewise, D2PL algorithm has two phases: the growing phase and the shrinking phase. The growth phase is the phase during which a transaction acquires locks and enters the critical section (accesses the shared resource or data), and the shrinking phase is that during which it releases the locks and exits the critical section.

The complexity of the growing phase for the D2PL algorithm is:

- Time complexity: O(k *(m + n)), where k is the number of transactions, m is the number of locks requested and n the number of participant nodes. The growth phase consists of acquiring the locks necessary for the transaction. Each lock is acquired in constant time, so the complexity is linear with respect to the number of locks requested.
- Space complexity: O(k + m), where k is the number of transactions and m is the number of locks requested. The growth phase requires storing the acquired locks and pending transactions.

The complexity of the shrinking phase for the D2PL algorithm is:

- Time complexity: O(k*(m + n)), where k is the number of transactions, m is the number of locks held and n the number of participant nodes. The shrinking phase consists of releasing the locks held by the transaction. Each lock is released in constant time, so the complexity is linear with respect to the number of locks held.
- Space complexity: O(k + m), where k is the number of transactions and m is the number of locks held. The shrinking phase requires storing the held locks and pending transactions.

Since a node cannot execute more than one transaction at a given time, the term "*k*" in these complexities models presented in Table 5 bellow, which is the number of transactions, symbolizes as well "*n*" the number of peers. Thus, the same Table 5, presents the models obtained after analysing the time and space complexity of the D2PL algorithm.

Table 5: Models of the time and space complexity of the D2PL algorithm.

Algorithm phases	Time complexity	Space complexity
Growing phase	O(k*(m + n))	O(n + m)
Shrinking phase	O(k*(m + n))	O(n + m)

It is important to note that complexity can vary depending on the specific implementation and system characteristics, such as network topology and lock management policy.

Factors influencing complexity:

- Number of concurrent transactions;
- Number of locks held per transaction;
- Lock management policy (e.g., FIFO, priority, etc.);
- Network topology and communication latency.

In general, the growing and the shrinking phases of the D2PL are considered to have reasonable complexity for medium-sized systems, but it can become a bottleneck for large systems or those with high concurrency, like P2P networks.

3) Efficiency comparison

Consider a P2P network consisting of a set P of peers whose cardinality, denoted #P ∈ [2, *n*], is determined randomly by the concurrent participants, selected from 12 scenarios. Let us assume *n* a random sample of participating peers. Let also *m = 10* be the number of locks held by each peer that has executed a transaction *n*. As the methodology for analysing the efficiency of algorithms aims to associate with each one a function *f(n)* or the number *t* which characterizes the execution time as a function of the input size *n*, Table 6 presents the results of the efficiency (of the running times in milliseconds (ms)) comparison between the new P2P-DMEA and the D2PL.

Table 6: Results of the efficiency analysis based on the running time complexity models.

Obs.	1	2	3	4	5	6	7	8	9	10	11	12
Number of participating peers (*n*)	11	23	35	47	59	71	83	95	107	119	131	143
P2P-DMEA token getting phase (in ms)	11	23	35	47	59	71	83	95	107	119	131	143
P2P-DMEA token releasing phase (in ms)	121	529	1225	2209	3481	5041	6889	9025	11449	14161	17161	20449
Number of transactions	11	23	35	47	59	71	83	95	107	119	131	143
Number of locks requested by a transaction	110	230	350	470	590	710	830	950	1070	1190	1310	1430
Number of locks held by a transaction	110	230	350	470	590	710	830	950	1070	1190	1310	1430
D2PL Growing phase (in ms)	1221	5313	12285	22137	34869	50481	68973	90345	114597	141729	171741	204633
D2PL Shrinking phase (in ms)	1331	5819	13475	24299	38291	55451	75779	99275	125939	155771	188771	224939

The results presented in Table 6 above, successively for the two approaches, the new P2P-DMEA and D2PL algorithm, lead to the following observations:

- Between the time complexity model of the token getting phase of P2P-DMEA and the growing phase of the D2PL algorithm, both phases during which a peer excludes others to access the shared resource, there is a clear

difference in performance. Looking at the results in the Table 6, one can realise that the token getting phase of P2P-DMEA is more efficient than the growing phase of the D2PL algorithm. The same is observable in the graph (*left*) of Fig. 5. The curve (in blue) of the token getting phase of P2P-DMEA is almost parallel to the abscess axis, meaning it has a very shallow slope, while that (in yellow) of the growing phase of the D2PL algorithm is increasing; Hence its steep slope.

- Since the P2P-DMEA token release phase model has quadratic time complexity, its execution time is much higher but it does exceed that of the D2PL algorithm's shrinking phase. The result presented in the Table 6 proves this, and it is also observable in the graph (*right*) of Fig. 5. The curve (in blue) of the P2P-DMEA token release phase has a slow exponential growth compared to the curve (in yellow) of the D2PL algorithm's shrinking phase which shows a much steeper growth slope. Therefore, the P2P-DMEA token release phase is more efficient than the D2PL algorithm's shrinking phase.

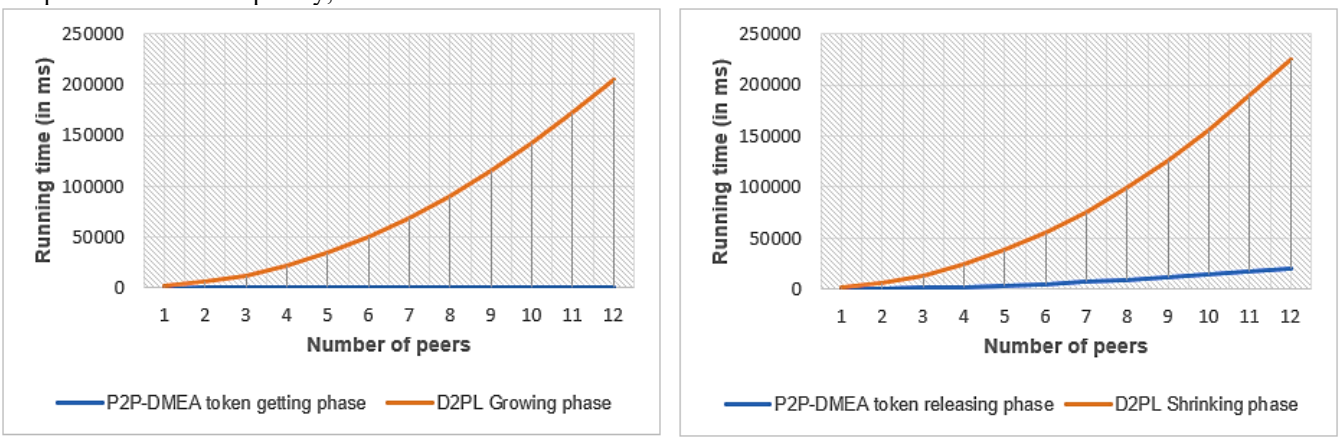


Fig. 5. Efficiency comparison : P2P-DMEA token getting vs D2PL Growing phase (*left*) and P2P-DMEA token release vs D2PL shringi phase (*right*).

In conclusion, we can therefore state that the D2PL algorithm has a higher time complexity than the P2P-DMEA algorithm. However, it is important to note that the D2PL algorithm is more robust and reliable than the P2P-DMEA algorithm because it uses a locking mechanism to ensure data consistency. However, it remains inefficient in the context of large-scale networks such as the P2P architecture. Thus, the P2P-DMEA algorithm outperforms the D2PL algorithm in terms of time complexity, although the D2PL algorithm is more robust and reliable on small networks.

V. Conclusion

This article proposes a new algorithm for serializing update transactions in a replicated database environment over a decentralized P2P architecture. Performing data replication on a decentralized P2P network, frequently updates made by refresh transactions from different peers reach a destination peer at the same time and are performed concurrently. This illustrated scenario is a problem that break the insurance of the consistency and availability among replicas. To solve this problem and to achieve the goal of this research, the algorithmic method combining with the token ring network topology allowed creating a new circulating token algorithm adapted on the P2P system. Based on the analysis of its theoretical temporal complexity, a comparative study was carried out between the new approach and the existing approach. This study revealed that the new approach effectiveness was promising.

However, our approach did not take into account the aspect that when a peer transfers the token to another on the requestor list, they should normally pass the list of other requesters remaining with the new beneficiary; in such a way that it too when it surrenders, it chooses the one which fulfils the conditions.

Declarations

Funding: The author did not receive support from any organization for the submitted work.

Competing interests: The author has no competing interests to declare that are relevant to the content of this article.

Availability of data and materials: Not applicable.

References

[1] M. Bakhta and B. Ghalem, "Managing Data Replication and Placement Based on Availability", Procedia-AASRI Conference on Parallel and Distributed Computing Systems, Publisher: Elsivier, Vol. 5, pp. 147 – 155, 2013.

[2] A. Souri, S. Pashazadeh and A. Navin, "Consistency of data replication protocols in database systems: A review", International Journal on Information Theory (IJIT), Vol. 3, pp.19-32, 2014.

[3] Z. Ting and W. Yu, "Database Replication Technology having high Consistency Requirements", IEEE Third International Conference on Information Science and Technology, March 2013.

[4] Nicoleta I. and Magdalena, "The Replication Technology in E-learning Systems", Procedia - Social and Behavioral Sciences, Publisher: Elsivier, Vol. 28, pp. 231 – 235, 2011.

[5] I. Filip, C.Vasar and R. Robu, "Considerations about an Oracle Database Multi-Master Replication", IEEE 5th International Symposium on Applied Computational Intelligence and Informatics, May 2009.

[6] Q. Vu, M. Lupu and C. Ooi, Peer-to-Peer Computing - Principles and Applications, Publisher: Springer, ISBN: 978-3-642-03513-5, January 2010.

[7] M. Wiesmann and al., "Understanding Replication in Databases and Distributed Systems", IEEE Xplore, 2010.

[8] M. T. Özsu and P. Valduriez, "Principles of Distributed Database Systems (3rd ed.)", New York, USA: © Springer Science+Business Media, LLC, 2011.

[9] A. George and C. Balakrishnan, "An optimized strategy for replication in peer-to-peer distributed databases", IEEE Xplore, 2012

[10] T. Zhang, “A Novel Replication Model with Enhanced Data Availability in P2P Platforms”, International Journal of Grid and Distributed Computing, Vol. 9, No. 4, pp.151-160, 2016.

[11] A. Srivastava, U. Shankar and S. Kumar, “Transaction Management in Homogenous Distributed Real-Time Replicated Database Systems”, International Journal of Advanced Research in Computer Science and Software Engineering, Vol. 2, pp. 2277 128X, 2012.

[12] M. Kaur and H. Kaur, “Concurrency Control in Distributed Database System”, International Journal of Advanced Research in Computer Science and Software Engineering, Vol. 3, pp. 2277 128X, July 2013.

[13] V. Sinde and P. Aware, “Concurrency control in distributed database systems”, International Journal for Research in Engineering Application & Management (IJREAM), Vol. 1, pp. 94-9150, 2016.

[14] H. Joshi and G.R. Bamnote, “Distributed Database: A Survey”, International Journal Of Computer Science And Applications, Vol. 6, pp. 0974-1011, 2013.

[15] I. Singh and S. Singh, Distributed Database Systems : Principles, Algorithms and Systems, New-Delhi, India : Khanna Book Publishing, Co. (P) Ltd, 2015.

[16] S. Ceri and G. Pelagatti, “Distributed Databases, Principles and systems”, New York, USA: © McGraw-Hill, Inc. 2008.

[17] M. K. Sah, V. Kumar and A. Tiwari, “Security and Concurrency Control in Distributed Database System”, International Journal of scientific research and management, (IJSRM), Vol. 2, pp. 2321-3418, 2014.

[18] A. Gupta and Y. Gore, “Concurrency Control and Security Issue in Distributed Database System”. International Journal of Engineering Development and Research, Vol. 4, pp. 177-181, 2016.

[19] S. L. Hernane, « Modèles et algorithmes de partage de données cohérents pour le calcul parallèle distribuée à haut débit », Thèse (en co-tutelle) présenté et soutenu publiquement pour l’obtention du doctorat de l'Université de Lorraine (spécialité informatique) et de l'Université des Sciences et de la Technologie d'Oran (USTO) (spécialité informatique), pp. 24-37, Juin, 2013.

[20] E. W. Dijkstra, “Solution of a Problem in Concurrent Programming Control”. Communications of the ACM, Vol. 8, No. 9, p. 569. September, 1965.

[21] E. W. Dijkstra, “Cooperating Sequential Processes’. In: Hansen, P.B. (eds) The Origin of Concurrent Programming. Springer, New York, NY. https://doi.org/10.1007/978-1-4757-3472-0_2, 1968.

[22] E. K. Kituta, S. Kant, R. Agarwal, “A systematic review on distributed databases systems and their techniques”, Journal of Theoretical and Applied Information Technology Vol. 96, No. 1, pp. 236-266, 2019.

[23] M. Raynal, “A simple taxonomy for distributed mutual exclusion algorithms”. SIGOPS Oper. Syst. Rev., Vol. 25, No. 2, pp. 47–50, avril 1991.

[24] J. Coulouris, J. Dollimore, T. Kindberg and G. Blair. “Distributed Systems, Concepts and design”, India: Pearson, pp. 629-720, 2012.

[25] D. K. Ajay, and M. Singhal, “Distributed Computing Principles, Algorithms, and Systems”, Cambridge, United Kingdom: Cambridge University Press, pp. 335-336, 2008.

[26] A. S.Tanenbaum and M. V. Steen. “Distributed systems: Principles and paradigms, Second Edition”, Vrije University, Amsterdam, Neterland: Pearson Prentice Hall, pp. 271-279, 2007.

[27] Y. Ding, J. Wang and K. P. Srimani. “Self-Stabilizing Master-Slave Token Circulation Algorithm in Undirected Rings and Unicyclic Graphs of Arbitrary Size and Their Orientations”, *International Journal of Networking and Computing*, Vol. 4, No. 1, pp. 42-52, 2014.

T. G. Michael and T. Roberto, “Algorithm design: Foundations, Analysis and Internet Examples (2nd ed.)”, New-Delhi, India, John Wiley & Sons, Inc., pp. 5-12, 2011.